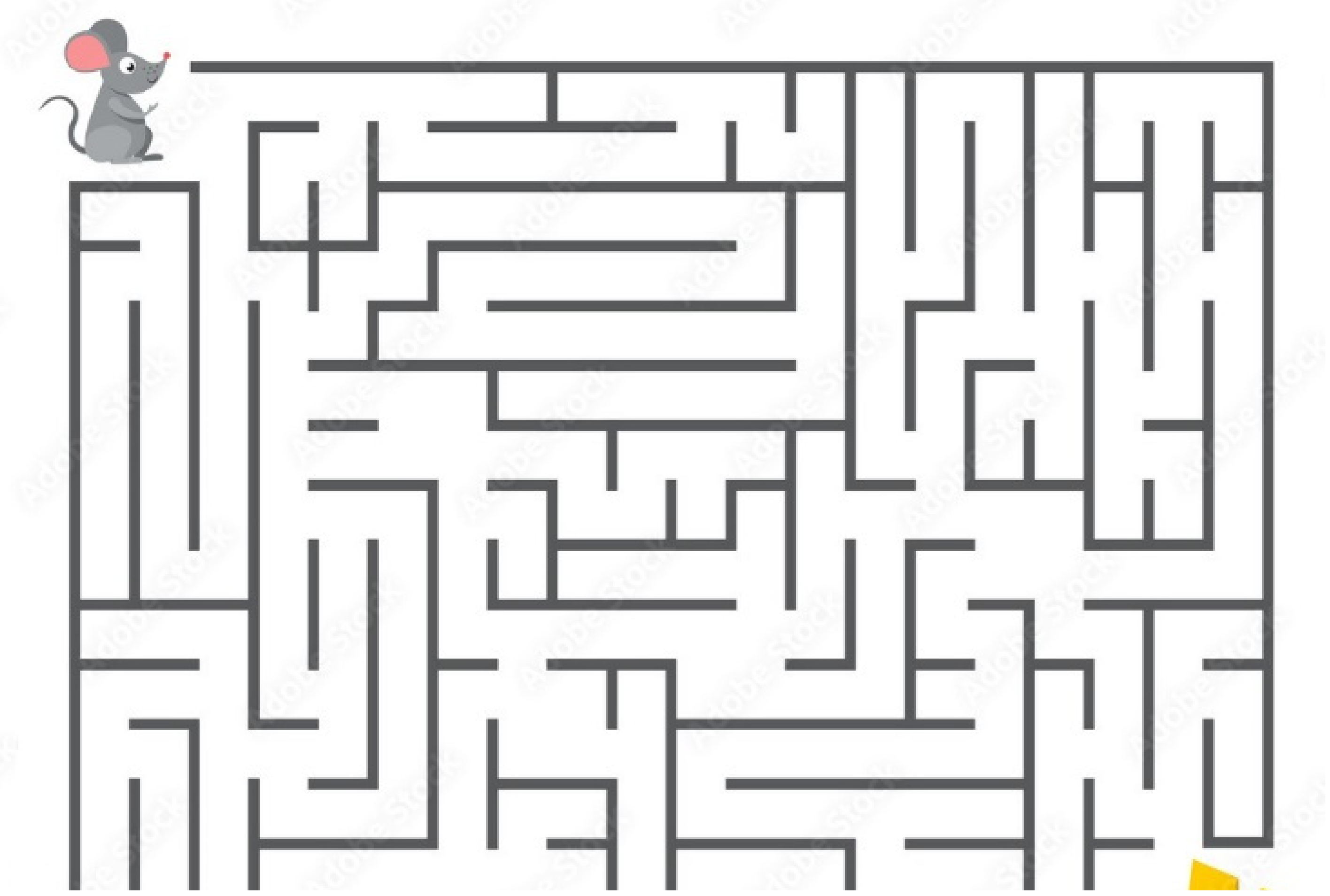
<u>DESIGN AND ANALYSIS OF ALGORITHM</u>

PROJECT BASED ON

"RAT IN MAZE PROBLEM USING BACKTRACKING ALGORITHM"

INTRODUCTION

- ***You may remember the maze game from childhood where a player starts from one place and ends up at another destination via a series of steps. This game is also known as the rat maze problem.***

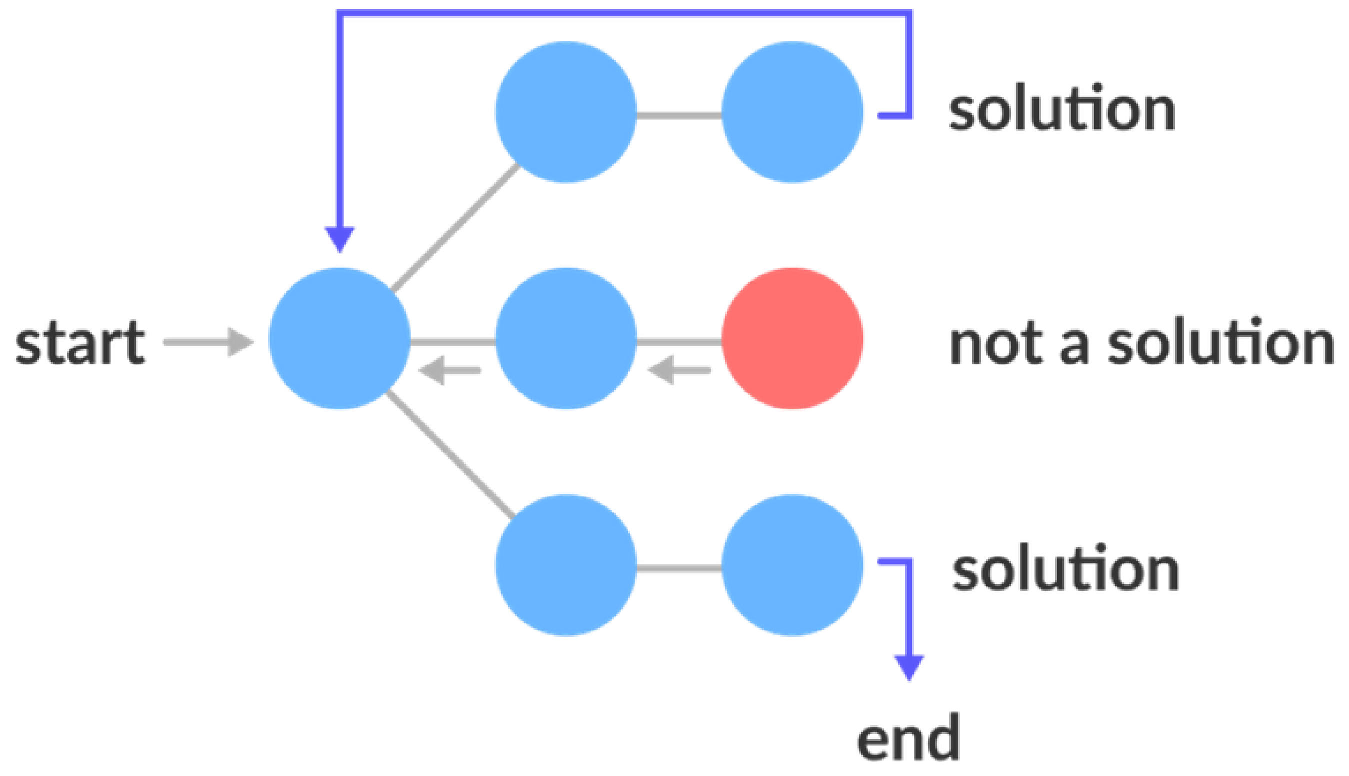


PROBLEM STATEMENT AND OBJECTIVES

- **In this problem, there is a given maze of size N x N. The source and the destination location is top-left cell and bottom right cell respectively. Some cells are valid to move and some cells are blocked. If one rat starts moving from start vertex to destination vertex, we have to find that is there any way to complete the path, if it is possible then mark the correct path for the rat.**
- **The maze is given using a binary matrix, where it is marked with 1, it is a valid path, otherwise 0 for a blocked cell.**
- **NOTE: The rat can only move in two directions, either to the right or to the down.**

Backtracking Algorithm

- Backtracking is an algorithmic-technique for solving problems recursively by trying to build a solution incrementally. Solving one piece at a time, and removing those solutions that fail to satisfy the constraints of the problem at any point of time (by time, here, is referred to the time elapsed till reaching any level of the search tree) is the process of backtracking.



- The term backtracking suggests that if the current solution is not suitable, then backtrack and try other solutions. Thus, recursion is used in this approach.

- ***State Space Tree:***

- **A space state tree is a tree representing all the possible states (solution or nonsolution) of the problem from the root as an initial state to the leaf as a terminal state.**

- ***Backtracking Algorithm:***

- **Backtrack(x)**
- **if x is not a solution**
- **return false**
- **if x is a new solution**
- **add to list of solutions**
- **backtrack(expand x)**

APPROACH AND ALGORITHM FOR THE PROBLEM

- **Form a recursive function, which will follow a path and check if the path reaches the destination or not. If the path does not reach the destination then backtrack and try other paths.**
- Create a solution matrix, initially filled with 0's.
- Create a recursive function, which takes initial matrix, output matrix and position of rat (i, j).
- if the position is out of the matrix or the position is not valid then return.
- Mark the position output[i][j] as 1 and check if the current position is destination or not. If destination is reached print the output matrix and return.
- Recursively call for position (i-1,j), (I,j-1), (i+1, j) and (i, j+1).
- Unmark position (i, j), i.e output[i][j] = 0.

- **isValid(x, y)**
- **Input:** x and y point in the maze.
- **Output:** True if the (x,y) place is valid, otherwise false.
- Begin
- **if x and y are in range and (x,y) place is not blocked, then**
- **return true**
- **return false**
- **End**

Solve RatMaze(x, y)

Input **− The starting point x and y.**
Output **− The path to follow by the rat to reach the destination, otherwise false.**

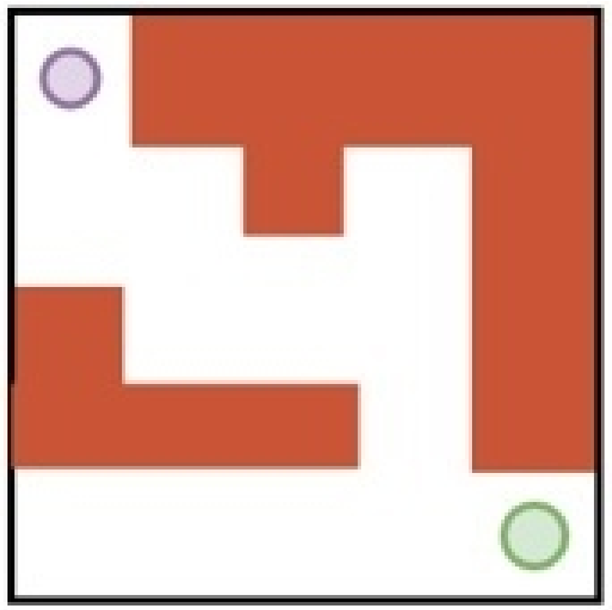
```
Begin
   if (x,y) is the bottom right corner, then
      mark the place as 1
      return true
   if isValidPlace(x, y) = true, then
      mark (x, y) place as 1
      if solveRatMaze(x+1, y) = true, then //for forward movement
         return true
      if solveRatMaze(x, y+1) = true, then //for down movement
         return true
      mark (x,y) as 0 when backtracks
      return false
   return false
End
```

SAMPLE PROBLEM

Input:

- This algorithm will take the maze as a matrix.
- In the matrix, the value 1 indicates the free space and 0 indicates the wall or blocked area.
- In this diagram, the top-left circle indicates the starting point and the bottom-right circle indicates the ending point.

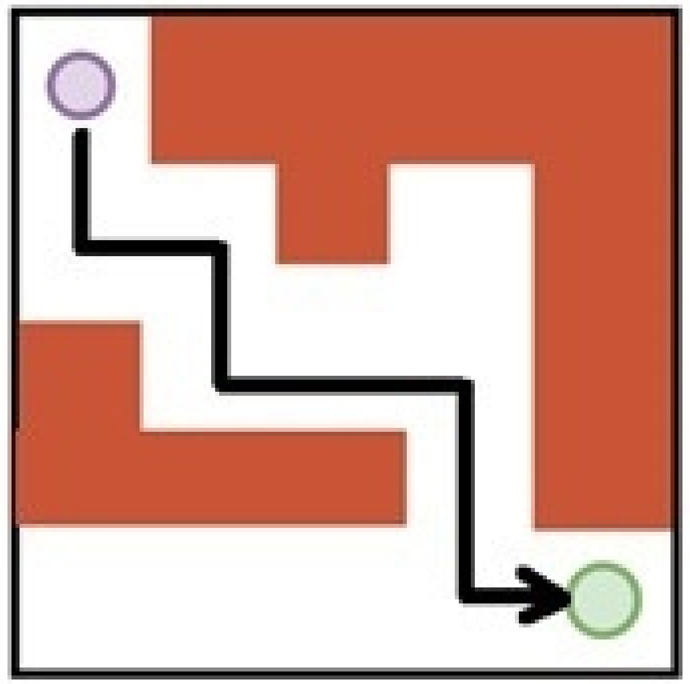
$$\begin{matrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \end{matrix}$$



OUTPUT:

- It will display a matrix. From that matrix, we can find the path of the rat to reach the destination point.

1	0	0	0	0
1	1	0	0	0
0	1	1	1	0
0	0	0	1	0
0	0	0	1	1



IMPLEMENTATION

```python

def is_valid(maze, row, col):
    """
    Checks if the given cell is within maze boundaries and unblocked.
    """
    n = len(maze)
    return 0 <= row < n and 0 <= col < n and maze[row][col] == 1

2 usages
def find_path(maze, row, col, solution):
    """
    Implements the backtracking algorithm to find a path in the maze.
    """
    if row == len(maze) - 1 and col == len(maze[0]) - 1:
        # Reached destination, mark the path and return True
        solution[row][col] = 1
        return True

    # Check if the current cell is within maze boundaries and unblocked
    if 0 <= row < len(maze) and 0 <= col < len(maze[0]) and maze[row][col] == 1 and solution[row][col] == 0:
        # Mark current cell as visited in the solution matrix
        solution[row][col] = 1

        # Try all possible directions (up, down, left, right)
        for dx, dy in [(0, 1), (1, 0), (0, -1), (-1, 0)]:
```

RAT IN MAZE.py

```python
            # Recursively call for the new position
            if find_path(maze, new_row, new_col, solution):
                return True  # Path found, return True

        # Backtrack if none of the directions lead to a solution
        solution[row][col] = 0
        return False
    else:
        return False

def solve_maze(maze):
    """
    Solves the maze using the backtracking algorithm.
    """
    n = len(maze)
    solution_matrix = [[0 for _ in range(n)] for _ in range(n)]  # Initialize solution matrix

    if find_path(maze, 0, 0, solution_matrix):
        print("Solution exists:")
        for row in solution_matrix:
            print(row)
    else:
        print("No solution exists")
```

RAT IN MAZE.py

```python
        print("No solution exists")


1 usage
def get_user_input():
    """
    Gets maze input from the user.
    """
    try:
        rows = int(input("Enter the number of rows in the maze: "))
        cols = int(input("Enter the number of columns in the maze: "))

        maze = []
        for i in range(rows):
            row_input = list(map(int, input(f"Enter row {i + 1} (0s and 1s separated by spaces): ").split())
            maze.append(row_input)

        return maze
    except ValueError:
        print("Invalid input. Please enter valid integers for rows and columns.")
        sys.exit(1)

if __name__ == "__main__":
    # Get maze input from the user
    user_maze = get_user_input()

    # Solve the maze and print the output
    solve_maze(user_maze)
```

```
PycharmProjects\AbhilashProject\.venv\Scripts\python.exe "C:\Users\ABHILASH\PycharmProjects\Abhilas
f rows in the maze: 5
f columns in the maze: 5
d 1s separated by spaces): 1 0 0 0 0
d 1s separated by spaces): 1 1 0 1 0
d 1s separated by spaces): 0 1 1 1 0
d 1s separated by spaces): 0 0 0 1 0
d 1s separated by spaces): 1 1 1 1 1



ith exit code 0
```

COMPLEXITY ANALYSIS

- ***Time Complexity:*** **O(2^(n^2)).**

  The recursion can run upper-bound 2^(n^2) times.

- ***Space Complexity:*** **O(n^2).**

  Output matrix is required so an extra space of size n*n is needed.

CONCLUSION

- **Using Backtracking algorithm , rat in maze problem can be solved easily as it's very intuitive to code, it is a step-by-step representation of a solution to a given problem, which is very easy to understand and it has got a definite procedure.**
- **A backtracking algorithm makes an effort to build a solution to a computational problem incrementally. Whenever the algorithm needs to choose between multiple alternatives to the next component of the solution, it simply tries all possible options recursively step-by-step.**

THANK YOU

TEAM MEMBERS :

- **DHARUN A - RA2211028010086**
- **ADITHYA P - RA2211028010075**
- **ABISHEK REDDY - RA2211028010092**